arXiv:submit/4735285 [cs.LG] 12 Feb 2023

Flag Aggregator: Scalable Distributed Training under Failures and Augmented Losses using Convex Optimization

Hamidreza Almasi **Harsh Mishra** **Balajee Vamanan** **Sathya N. Ravi**
Department of Computer Science
University of Illinois at Chicago
{halmas3, hmishr3, bvamanan, sathya}@uic.edu

Abstract

Modern ML applications increasingly rely on complex deep learning models and large datasets. There has been an exponential growth in the amount of computation needed to train the largest models. Therefore, to scale computation and data, these models are inevitably trained in a distributed manner in clusters of nodes, and their updates are aggregated before being applied to the model. However, a distributed setup is prone to byzantine failures of individual nodes, components, and software. With data augmentation added to these settings, there is a critical need for robust and efficient aggregation systems. We extend the current state-of-the-art aggregators and propose an optimization-based subspace estimator by modeling pairwise distances as quadratic functions by utilizing the recently introduced Flag Median problem. The estimator in our loss function favors the pairs that preserve the norm of the difference vector. We theoretically show that our approach enhances the robustness of state-of-the-art byzantine resilient aggregators. Also, we evaluate our method with different tasks in a distributed setup with a parameter server architecture and show its communication efficiency while maintaining similar accuracy. The code is publicly available at https://github.com/hamidralmasi/FlagAggregator

1 Introduction

How to Design Aggregators? We consider the problem of designing aggregation functions that can be written as optimization problems of the form,

$$\mathcal{A}(g_1, \ldots, g_p) \in \arg\min_{Y \in C} A_{g_1,\ldots,g_p}(Y), \tag{1}$$

where $\{g_i\}_{i=1}^p \subseteq \mathbb{R}^n$ are given estimates of an unknown summary statistic Y^*. If we choose A to be a quadratic function that decomposes over g_i's, and $C = \mathbb{R}^n$, then we can see $\mathcal{A}$ is simply the standard mean operator. There is a mature literature of studying such functions for various scientific computing applications [1]. More recently, from the machine learning standpoint there has been a plethora of work [2, 3, 4, 5] on designing provably robust aggregators $\mathcal{A}$ for mean estimation tasks under various technical assumptions on the distribution of g_i or its moments $\mathbb{E}(g_i^{\mathrm{m}})$. While these developments assert that efficient estimation is possible even in the presence of noise, they use complex convex optimization solvers as a *black box*, thus impractical in large scale settings that we encounter in practice. In particular, the practical implementation aspects and more importantly, the total time complexity remains unclear to a large extent.

To make the discussion concrete, consider training a model with a large dataset such as ImageNet-1K [6] or its augmented version which would require data to be distributed over p workers and uses back propagation. Indeed, in this case, g_i's are typically the gradients computed by individual workers at

Preprint. Under review.

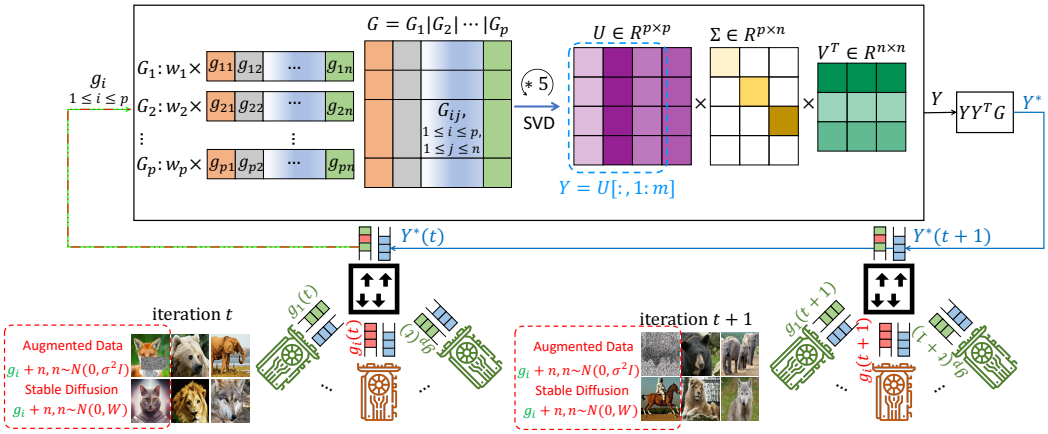


Figure 1: Robust gradient aggregation in our distributed training framework. In our applications, each of the p workers provides gradients computed using a random sample obtained from given training data, derived synthetic data from off-the-shelf Diffusion models, and random noise in each iteration. Our Flag Aggregator (FA) removes high frequency noise components by using few rounds of Singular Value Decomposition of the concatenated Gradient Matrix G, and provides new update Y^*.

each iteration, and so we may design $\mathcal{A}$ such that the total iteration or communication complexity is decreased as much as possible. In settings where the training objective is convex, the convergence and generalization properties of distributed optimization can be achieved by defining $\mathcal{A}$ as a weighted combination of gradients facilitated by a simple consensus matrix, even if some g_i's are noisy [7, 8]. In stark contrast, there is *no* existing gradient aggregation rule using a linear combination of the gradients sent by workers that can tolerate even a single byzantine worker in nonconvex settings [9]. Therefore, designing robust and efficient aggregation schemes is still open for distributed training.

As we distribute data among workers, hardware and software failures in workers may cause them to send erroneous gradients, which could mislead the model in a completely different direction compared to the gradients computed by correctly functioning workers [10]. While the probability of failure increases with scale, individual component failures (e.g., Memory [11], Disk [12], Network [13]), as well as software failures (e.g., bugs, security flaws), are becoming increasingly common [14], as these failures were observed not only in large datacenters but also in HPC clusters [15, 16]. Reliability theory is the standard paradigm to analyze such failures, see Chapter 9 in [17]. For our purposes, we just note that distribution of number of total systems failures does not factorize over workers (two workers installed at similar times usually fail together), and so the total noise in gradients is *dependent* – the key challenge for large scale training. Moreover, even if there is no issues with hardware infrastructure, nonlinear data augmentation routines used to inpaint and/or fill-in missing information (say at blurred pixels) in training data is a possible source of dependent noise. Most existing open source implementations of $\mathcal{A}$ rely just on (functions of) pairwise distances to filter gradients from workers. While this may be a good strategy when the noise in samples/gradients is independent, these methods are suboptimal when the noise is dependent. Moreover, choosing discrete hyperparameters such as number of neighbors is impractical in our use cases since they hamper convergence of the overall training procedure.

In this paper, we explore the problem of designing communication-efficient aggregation to handle *Byzantine* faults due to random worker failures, and nonlinear data augmentation. In Figure 1, our proposed Flag Aggregator (FA) is computed using gradients from different workers (i.e., GPUs) – the red colored workers are considered Byzantine either because the gradients get corrupted due to random noise, or modified due to nonlinear data augmentation routines or stable diffusion during training. We present a new optimization based formulation for generalized gradient aggregation purposes in the context of distributed training of deep learning architectures, as shown in Figure 1. We then argue with extensive empirical results that it is possible to realize benefits of using FA in Distributed ML setups.

Summary of our Contributions. From the **theoretical** perspective, we analyze the tractability of our formulation using techniques from Convex Optimization. Specifically, we show that our formulation has tractable semidefinite programming relaxations that can be solved efficiently using off-the-shelf solvers for small amount of noise. For larger noise levels, we show that a factored form of our relaxation can be solved efficiently using first order reweighing methods. **Experimentally**, our results show that nonlinear data augmentation schemes that can be efficiently handled while training using our aggregation scheme. In practice, we achieve a *significantly* ($\approx 20\%$) better accuracy on standard datasets. Our **implementation** provides significant benefits for minimizing communication complexity under various noise settings with our new aggregation function on many use cases. We also provide an in-depth discussion of the programmability aspects of our FA for training large-scale models in customized distributed ML settings.

2 Robust Aggregators as Orthogonality Constrained Optimization

Basic intuition from Linear Algebra. If $g_i \in \mathbb{R}^n$ is the gradient given by worker i, and $Y \in \mathbb{R}^{n\times m}$ is a subspace that gradients could live in, then each column in Y as a function can act on g_i, i.e., each coordinate of $(Y^T g)_j$ for $1 \leq j \leq m$ is the application of m basis or columns in Y on g. Recall that by definition of dot product, we have that if $Y_{:,j} \perp x$, then $(Y^T g)_j$ will be close to zero. Equivalently, if $g \in \text{span}(Y)$, then $(Y^T g)^T Y^T g$ will be bounded away from zero.

Assuming that $G \in \mathbb{R}^{n\times p}$ is the gradient matrix of p workers, $YY^T G \in \mathbb{R}^{n\times p}$ is the reconstruction of X using Y as basis. The weight of worker i can be some norm of ith column of $Y^T G$. l_∞ of $Y^T g_i$ tells us if there is a basis in Y such that $Y \not\perp g_i$. The average over columns of $YY^T G$ would give the final gradient for update.

Flag Aggregator for Distributed Optimization.

Flag Median is an extension of the median in one-dimensional vector spaces to ***different*** finite dimensional subspaces using the so-called chordal distance between them. From the optimization point of view, this makes the computational problem easier because *squared* chordal distance is smooth everywhere whereas other natural distances such as geodesic may not be, thus slow convergence.

Based on the Flag Median estimator for subspaces, we propose an optimization based subspace estimator Y^* by modeling pairwise distances as quadratic functions. We formulate these quadratic functions of Y using trace operators $\text{tr}(\cdot)$ to define our proposed Flag Aggregator (FA) as the solution of the following constrained optimization problem:

$$\min_{Y:Y^TY=I} A(Y) := \frac{1}{p}\sum_{i=1}^{p}\sqrt{\left(1-\frac{\text{tr}\left(Y^T g_i g_i^T Y\right)}{\|g_i\|_2^2}\right)} + \frac{\lambda}{p(p-1)}\sum_{i,j=1}^{p}\sqrt{\left(1-\frac{\text{tr}(Y^T(g_i-g_j)\left(g_i-g_j\right)^T Y)}{D_{ij}^2}\right)}, \tag{2}$$

where $D_{ij} = \|g_i - g_j\|_2^2$ denotes the distance between gradient vectors g_i, g_j from workers i, j and $\lambda > 0$ is a regularization hyperparameter.

our hypothesis is that the true gradient lies in the subspace spanned by the clean gradients which is nicely captured by the optimization-based FM framework in (1) by Y. In (practical) distributed settings, the quality (or noise level) of gradients in each worker may be different, **and/or** each worker may use a different batch size. In such cases, handcrafted aggregation schemes may be difficult to maintain, and fine-tune. Our FM-based FA scheme can easily handle these scenarios – we can simply reweigh gradients of worker i according to its noise level before projecting onto real Stiefel manifold, **and/or** use $g_i \in \mathbb{R}^{n\times b_i}$ where b_i is the batch size of $i-$th worker in formulation (2).

Intuitively, the pairwise terms in our loss function (2) favors subspace Y that also reconstructs the pairwise vectors $g_i - g_j$ that are close to each other. That is, if D_{ij} is small, an arbitrary (feasible) Y such that $Y^T Y = I$ – say, sampled uniformly at random from the Stiefel Manifold, – will have a value $tr(Y^T(g_i - g_j)(g_i - g_j)^T Y) \approx m$, thus making the loss value high. Conversely, if we can find Y^* such that $tr(Y^{*T}(g_i - g_j)(g_i - g_j)^T Y^*) \approx D_{ij}$ – that is, Y^* preserves the norm of the vector $g_i - g_j$, then the ratio will be ≈ 1, thus achieving a smaller loss value associated with that term.

Note that when $\|g_i\|_2^2 = 1$ are unit vectors, and $\lambda = 0$, our formulation coincides with the recently introduced Flag Median estimator as in [18]. On the other hand, when we set $\lambda = \Theta(p^2)$, that is, the pairwise terms dominate the objective function in (2). Hence, λ regularizes optimal solutions Y^* of (2) to contain g_i's with low pairwise distance in its span – similar to spirit to AggregaThor in [19]. For each worker i, by using the *smoothed* softmax score $\tilde{D}_{ij} \in (0, 1)$ such that $\sum_j \tilde{D}_{ij} = 1$ instead of the distances themselves in the denominator, we can specify the number of Byzantine workers in the formulation.

2.1 Tractability of Computing Flag Aggregators

In this section, we characterize the computational complexity of solving the flag aggregation problem via IRLS type schemes using results from convex optimization. First, we present a tight convex relaxation of the Flag Median problem by considering it as an instantiation of rank constrained optimization problem. We then show that we can represent our convex relaxation as a Second Order Cone Program which can be solved off-the-shelf solvers [20]. Second, we argue that approximately solving such rank constrained problems in the factored space is an effective strategy using new results from [21] which builds on asymptotic convergence in [22]. Our results highlight that the Flag Median problem can be approximately solved using smooth optimization techniques, thus explaining the practical success of a IRLS type iterative solver.

Interpreting Flag Aggregator (2) in the Case $m = 1$. We first present a convex reformulation of the Flag Aggregator problem (2) in the case when the number of subspaces (or columns) is equal to 1. To make the exposition easier, we will also assume that $\lambda = 0$. With these assumptions, and using the fact that $\|y\|_2 = 1$, each term in the objective function of our FA aggregator in (2) can be rewritten as,

$$\sqrt{(1 - (y^T \tilde{g}_i))^2)} = \sqrt{y^T \left(I - \tilde{g}_i \tilde{g}_i^T\right) y} = \|\tilde{B}_i y\|_2, \tag{3}$$

where we use the notation $\tilde{g}_i = g_i / \|g_i\|$ to denote the normalized worker gradients, $I \in \mathbb{R}^{n \times n}$ is the $n \times n$ identity matrix, and $\tilde{B}_i$ is the square root of the matrix $I - \tilde{g}_i \tilde{g}_i^T$. Observe that we can rewrite all the terms in (2) in a similar fashion as in (3). Furthermore, by relaxing the feasible set to the $n-$Ball given by $\{y \in \mathbb{R}^n : \|y\|_2 \leq 1\}$, we obtain a Second Order Cone Programming (SOCP) relaxation of our FA problem in (2). SOCP problems can be solved using off-the-shelf packages with open source optimization solvers for gradient aggregation purposes in small scale settings, that is, when the number of parameters $n \approx 10^4$ [23, 24, 25]. Our convex reformulation immediately yields insights on why reweighing type algorithm that was proposed in [18] works well in practice – for example see Section 3 in [26] in which various smoothing functions similar to the Flag Median (square) based smoothing are listed are options. More generally, our SOCP relaxation shows that if the smoothed version can be solved in closed form (or efficiently), then a reweighing based algorithm can be safely considered a viable candidate for aggregation purposes.

Tractable Reformulations in Higher Dimensions for Aggregation Purposes. Note that for any feasible Y such that $Y^T Y = I$, we have that the $\text{tr}(Y) = m$. Using this, we can rewrite each term in the objective function of our FA aggregator in (2) as,

$$\sqrt{\text{tr}\left(Y^T \left(\frac{I}{m} - \frac{g_i g_i^T}{\|g_i\|_2^2}\right) Y\right)} = \sqrt{\text{tr}\left(Y^T M_i Y\right)}, \tag{4}$$

where $M_i = M_i^T, i = 1, ..., p$ is symmetric matrix with *at most* one negative eigenvalue. Optimization problems involving quadratic functions with negative eigenvalues can be solved globally, in some cases [27, 28]. We consider methods that can efficiently (say in polynomial running time in n) provide solutions that are locally optimal. In order to do so, we consider the Semi Definite programming relaxation obtained by introducing matrix $Z \succeq 0 \in \mathbb{R}^{nm \times nm}$ to represent the term YY^T, constrained to be rank one, and such that $\text{tr}(Z) = m$. By using $\textbf{vec}(Y) \in \mathbb{R}^{nm}$ to denote the vector obtained by stacking columns of Z, when $m > 1$, we obtain a trace norm constrained SOCP. Importantly, objective function can be written as a sum of terms of the form,

$$\sqrt{\textbf{vec}(Y)^T \left(I \otimes M_i\right) \textbf{vec}(Y)} = \sqrt{\text{tr}\left(Z^T \left(I \otimes M_i\right)\right)}, \tag{5}$$

where $\otimes$ denotes the usual tensor (or Kronecker) product between matrices.

Properties of lifted formulation in (5). There are some advantages to the loss function as specified in our reformulation (5). First, note that then our relaxation coincides with the usual trace norm based Goemans-Williamson relaxation used for solving Max-Cut problem with approximation guarantees [29]. Albeit, our objective function is not linear, and to our knowledge, it is not trivial to extend the results to nonlinear cases such as ours in (5). Moreover, even when $M_i \succeq 0$, the $\sqrt{\cdot}$ makes the relaxation *nonconvex*, so it is not possible to use off-the-shelf disciplined convex programming software packages such as CVXPy [30, 31]. Our key observation is that away from 0, $\sqrt{\cdot}$ is a *differentiable* function. Hence, the objective function in (5) is differentiable with respect to Z.

2.2 Solving Flag Aggregation Efficiently

Now that we have a smooth reformulation of the aggregation problem that we would like to solve, it is tempting to solve it using first order methods. However, naively applying first order methods can lead to slow convergence, especially since the number of decision variables is now increased to m^2n^2. Standard projection oracles for trace norm require us to compute the full Singular Value Decomposition (SVD) of Z which becomes computationally expensive even for small values of $m, n \approx 10$.

Fortunately, recent results show that the factored form smooth SDPs can be solved in polynomial time using gradient based methods. That is, by setting $Z = \mathbf{vec}(Y)\mathbf{vec}(Y)^T$, and minimizing the loss functions $L_i(Y) = \sqrt{\mathbf{vec}(Y)^T\,(I \otimes M_i)\,\mathbf{vec}(Y)}$ with respect to Y, we have that the set of locally optimal points coincide, see [21]. Moreover, we have the following convergence result for first order methods like Gradient Descent that require SVD of $n \times p$ matrices:

Lemma 2.1. *If L_i are κ_i-smooth, with a η_i-lipschitz Hessian, then projected gradient descent with constant step size converges to a locally optimal solution to (5) in $\tilde{O}(\kappa/\epsilon^2)$ iterations where $0 \leq \epsilon \leq \kappa^2/\eta$ is a error tolerance parameter, $\kappa = \max_i \kappa_i$, and $\eta = \max_i \eta_i$.*

Above lemma 2.1 says that gradient descent will output an aggregation Y that satisfies second order sufficiency conditions with respect to smooth reformulated loss function in (5). All the terms inside $\tilde{O}$ in lemma 2.1 are logarithmic in dimensions m, n, lipschitz constant L, and accuracy parameter ϵ. We have included the proof of lemma 2.1 in Appendix A.

Remark 2.2. Note that the lipschitz constant κ of the overall objective function depends on M_i. That is, when M_i has negative eigenvalues, then κ can be high due to the square root function. We can consider three related ways to avoid this issue. First, we can choose a value $m' > m$ in our trace constraint such that $M_i \succeq 0$. Similarly, we can expand (5) (in $\sqrt{\cdot}$) as outer product of columns of Y suggesting that $\tilde{g}\tilde{g}^T$ term need to be normalized by m, thus making $M_i \succeq 0$. Secondly, we can consider adding a quadratic term such as $\|Y\|_{\text{Fro}}^2$ to make the function quadratic. This has the effect of decreasing κ and η of the objective function for optimization. Finally, we can use $m_i = \max(k_i, m)$ instead of $\min$ in defining the loss function as in [18] which would also make $M_i \succeq 0$.

Difference between Flag Median and our Flag Aggregator (FA). The main difference between flag median, and our aggregator is that our reformulation technique in (5) can be used even when $g_i \in \mathbb{R}^{k_i \times n} \notin \text{Gr}(k_i, n)$ are not subspaces as in [18]. Note that, if we have no wall clock time restrictions, we can use the closest orthogonal matrix of g_i computed via say QR factorization. However, in distributed training applications, this additional QR factorization can be an efficiency bottleneck. Moreover, since the aggregation is used as part of an iterative procedure, an approximate solution suffices, and is sometimes even preferred. The smooth reformulation in our Flag aggregation framework is easily applicable, as long as we have an upper bound the normalization factor of g_i which can be stored, and updated locally.

3 Experiments

In this section, we conduct experiments to test our proposed FA in the context of distributed training in two testbeds. First, to test the performance of our FA scheme solved using IRLS (Flag Mean) on standard Byzantine benchmarks. Then, to test the ability of existing state-of-the-art gradient aggregator we augment data via two techniques that can be implemented with standard differential equation solvers.

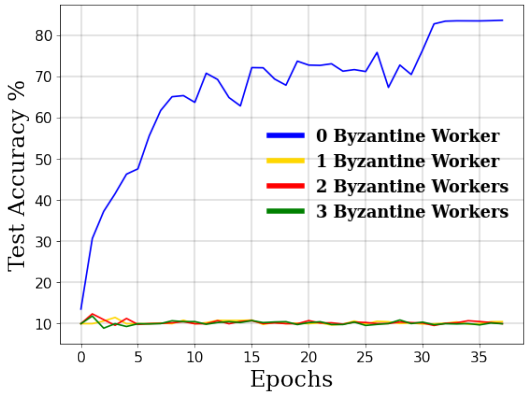
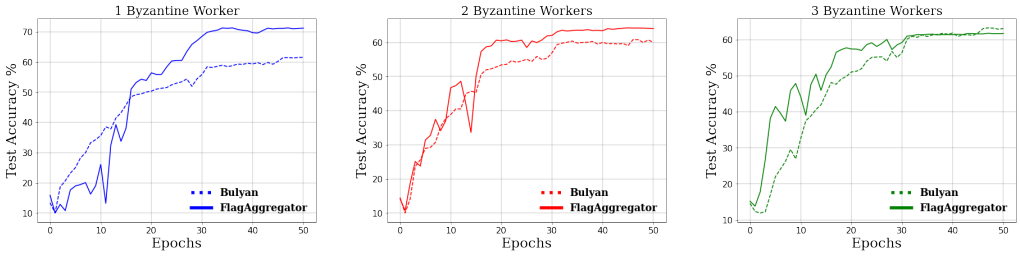


Figure 2: Tolerance to the number of byzantine workers for a non-robust aggregator (average).



(a) $f = 1$ (b) $f = 2$ (c) $f = 3$

Figure 3: Tolerance to the number of byzantine workers for robust aggregators for batch size 200.

3.1 Implementation

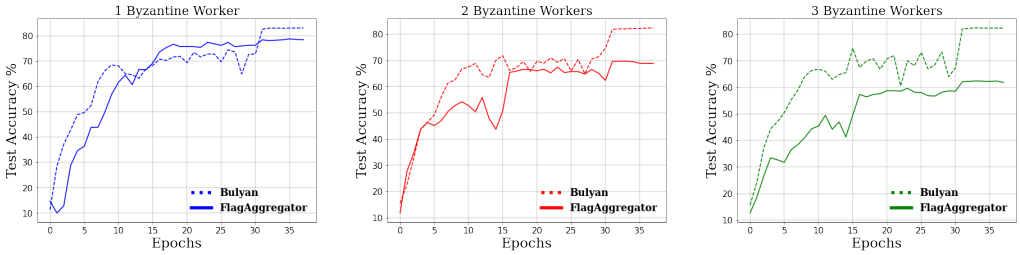
We implement flag aggregator in Pytorch [32], a popular ML framework which does not support byzantine resilience natively. We adopt the parameter server architecture and employ Pytorch's distributed RPC framework with TensorPipe backend for machine to machine communication. We extend Garfield's Pytorch library [33] with our flag aggregator and limit our IRLS convergence criteria to a small error, 10^{-10}, or 4 iterations of flag mean for SVD calculation.

3.2 Setup

Baseline: We compare flag aggregator to two baselines. The first baseline represents the performance of a distributed framework which is not byzantine resilient where we average the workers' gradients at the parameter server and send it back to them. For the second baseline we compare to Bulyan [34], a strong byzantine resilient gradient aggregation rule for $n_w \geq 4f + 3$ where n_w is the total number of workers and f is the number of byzantine workers. Bulyan is a two stage algorithm. In the first stage, a gradient aggregation rule R like coordinate-wise median [35] or Krum [9] is recursively used to select $\theta = n_w - 2f$ gradients. The process is to use R to select gradient vector g_i which is closest to R's output (e.g. for Krum, this would be the gradient with the top score, and hence the exact output of R). The chosen gradient is removed from the received set and added to the selection set S. This process is repeated if $|S| < \theta$. At the end of the first stage, S would contain the majority of non-byzantine gradients and the median of coordinates for each dimension of the gradient vector is always bounded by non-byzantine values. The second stage produces the resulting gradient, if $\beta = \theta - 2f$, each coordinate would be the average of β-nearest to the median coordinate of the θ gradients in S.

Accuracy. The fraction of correct predictions among all predictions, using the test dataset (top-1 cross-accuracy).

Testbed: We used 4 servers as our experimental platform. Each server has 2 Intel(R) Xeon(R) Gold 6240 18-core CPU @ 2.60GHz with Hyper-Threading and 384GB of RAM. Servers have a Tesla



(a) $f = 1$ (b) $f = 2$ (c) $f = 3$

Figure 4: Tolerance to the number of byzantine workers for robust aggregators for batch size 25.

V100 PCIe 32GB GPU and employ a Mellanox ConnectX-5 100Gbps NIC to connect to a switch. We use one of the servers as the parameter server and instantiate 15 workers on other servers, each hosting 5 worker nodes. We use Tensorpipe with InfiniBand channel to communicate among the servers.

Dataset and model: We focus on the image classification task since it is a widely used task for benchmarking in distributed training [36]. We train ResNet-18 [37] on CIFAR10 which has 60,000 32×32 color images in 10 classes. We use SGD as the optimizer, and cross-entropy to measure loss. The batch size in each worker is 200 unless otherwise stated. Except for the data augmentation experiment, in other experiments, byzantine workers send random gradients. We change the number of byzantine workers (f) from 1 to 3 in our experiments. We set $m = n_w - f - 2$ for FA throughout the experiments.

Evaluating Resilience against Nonlinear Data Augmentation: In order to induce byzantine behaviour in our workers we utilize ODE solvers to approximately solve 2 non-linear processes, Lotka Volterra [38] and Arnold's Cat Map [39], as augmentation methods.

In **Lotka Volterra**, we use the transformation of pixels as given by the following linear transformation in 2D:

$$(x, y) \rightarrow (\alpha x - \beta xy, \delta xy - \gamma y),$$

where α, β, γ and δ are hyperparameters. We choose them to be $\frac{2}{3}$, $\frac{4}{3}$, -1 and -1 respectively.

Second, we use a *nonsmooth* transformation called **Arnold's Cat Map** as a data augmentation scheme. Once again, the map can be specified using a two dimensional matrix as,

$$(x, y) \rightarrow \left(\frac{2x + y}{N}, \frac{x + y}{N} \right) \mod 1,$$

where $\mod$ represents the modulus operation, x and y are the coordinates or pixels of images and N is the height/width of images (assumed to be square). We also used a smooth approximation of the Cat Map obtained by approximating the $\mod$ function as,

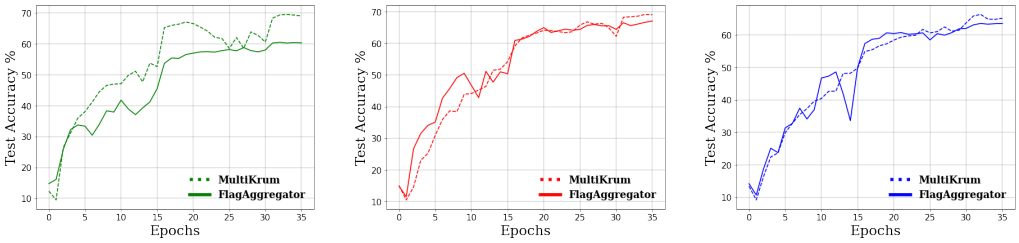
$$(x, y) \rightarrow \frac{1}{n} \left(\frac{2x + y}{(1 + \exp(-m \log(\alpha_1)}, \frac{x + y}{(1 + \exp(-m \log(\alpha_2)} \right),$$

where $\alpha_1 = \frac{2x+y}{n}$, $\alpha_2 = \frac{x+y}{n}$, and m is the degree of approximation, which we choose to be 0.95 in our data augmentation experiments.

How to perform nonlinear data augmentation? In all three cases, we used scipy's solve_ivp method to solve the differential equations, by using the 'LSODA' solver. In addition to the setup described above, we also added a varying level of Gaussian noise to each of the training images. All the images in the training set are randomly chosen to be augmented with varying noise levels of the above mentioned augmentation schemes. We have provided Python code that implements all our data augmentation schemes in the supplement zip folder.

3.3 Results

Tolerance to the number of byzantine workers: In this experiment we show the effect of byzantine behavior on the convergence of different gradient aggregation rules. Figure 2 shows that for a



(a) $n_w = 9$ (b) $n_w = 12$ (c) $n_w = 15$

Figure 5: The effect of adding more workers in the presence of fixed number of byzantine workers $f = 2$.

non-byzantine resilient rule, i.e. averaging, when there are no byzantine workers the system quickly converges to a state with high test accuracy. On the other hand, the presence of even a single byzantine worker has a fatal effect as shown in other works [9].

We repeat the same experiment with Bulyan and FA as robust aggregation rules. Figure 3 shows accuracy for different number of byzantine workers when the batch size is 200. For both Bulyan and FA, as the number of byzantine workers increases, filtering out the outliers becomes more challenging because the amount of noise increases. We noticed that after training for a certain number of epochs, the variance of gradients increased, and SGD started to diverge. To mitigate, we used a learning decay strategy where we decreased the learning rate by a factor of 0.2 every 15 epochs.

We also study the effect of different batch sizes in convergence. For this purpose, we compare FA to Bulyan with smaller batch size. Figure 4 shows the performance comparison when batch size is 25. **Our results indicate that, in cases when larger batch size is a training requirement, FA achieves a significantly better accuracy compared to the existing state of the art aggregators.** This may be useful in some large scale vision applications, see [40, 41] for more details.

Tolerance to communication loss: To analyze the effect of unreliable communication channels between the workers and the parameter server on convergence, we design an experiment where the link between some of the workers and the parameter server randomly drops a percentage of packets. In this experiment, we set the loss rate of one of the links to 10% which would essentially drop 10% of the packets going out of all the worker nodes that are hosted on that machine. In effect, there would be 5 byzantine workers with this behavior in our setting. Since $f = 5$ is higher than Bulyan's requirement $n_w \geq 4f + 3$, we compare FA to Multi-Krum for this experiment. The loss is introduced using the *netem* queuing discipline in Linux designed to emulate the properties of wide area networks where such a loss value is possible [42]. Here we set m for both FA and Multi-Krum to be $n_w - f - 2 = 8$.

The two main takeaways from our results in Figure 6 can be summarized as follows:

1. FA converges to a significantly higher accuracy than MultiKrum, and
2. Considering time-to-accuracy for comparison, FA reaches the same accuracy as Multi-Krum in a lesser total number of training iterations.

This implies that FA is more robust to slow underlying networks. The reason is that after packet loss, the parameter server waits up to a certain local timeout before requesting the lost gradients. This stalls convergence but the effect is less detrimental to FA since it is communication-efficient.

Analyzing the marginal utility of additional workers. To see the effect of adding more workers to a fixed number of byzantine workers, we ran experiments where we fixed f, and increased n. Our experimental results shown in Figure 5 indicate that our FA algorithm may possess strong resilience property as in MultiKrum [34] for reasonable choices of n.

The effect of having augmented data during training in byzantine workers: Figure 7 shows FA can handle nonlinear data augmentation in a much more stabler fashion. Please see Appendix B for exact details on the level of noise, and exact solver settings that were used to obtain augmented images. We provide more results in Appendix B.

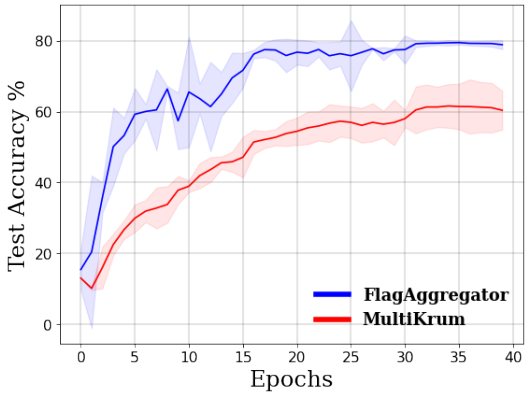


Figure 6: Tolerance to communication loss

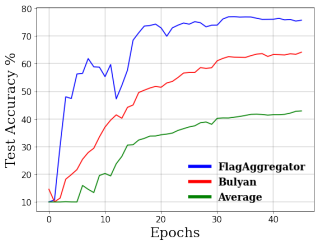


Figure 7: Accuracy of using augmented data in $f = 3$ workers

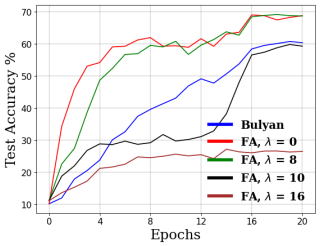


Figure 8: CIFAR10 with ResNet-18, $n = 7$, and $f = 1$

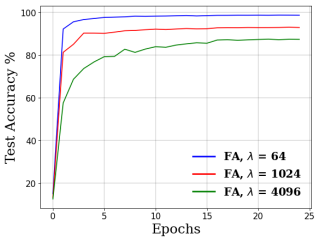


Figure 9: MNIST with $n = 60$ and $f = 14$

The effect of the regularization parameter in FA: The regularization parameter λ in FA provides flexibility in the loss function to cover aggregators that benefit from pairwise distances such as Bulyan and MultiKrum. To verify this experimentally, we change λ in Figure 8 to bridge the gap between Bulyan and we can see when FA improves or performs similarly to Bulyan for a range of λ.

Scaling out to real-world situations with more workers: In distributed ML, n and f are usually large. To test high-dimensional settings commonly dealt in Semantic Vision with our FA, we used ResNet-18. Now, to test the scalability of FA, we set up $n = 60$ workers ($f = 14$ byzantine) with MNIST dataset and a simple CNN (useful for simple detection). Figure 9 shows evidence that FA is feasible to larger setups for MNIST with $n = 60$, and $f = 14$, and also robust with respect to choice of λ during training.

4 Background and Related Work

Researchers have approached byzantine resilience problem from two main directions. In the first class of works, techniques such as geometric median and majority voting try to perform robust aggregation [19], [43], [9]. The other class of works uses redundancy and assigns each worker redundant gradient computation tasks [44], [45].

From another aspect, robustness can be provided in two levels. In weak byzantine resilience methods such as Coordinate-wise median [35] and Krum [9], the learning is guaranteed to converge. In strong byzantine resilience, the learning converges to a state as the system would converge in case no byzantine worker existed Draco [45] and Bulyan [34]. Convergence analysis of iterated reweighing type algorithms have been done for specific problem classes. For example, [46, 47] show that when IRLS is applied for sparse regression tasks, the iterates can converge linearly. Convergence analysis of matrix factorization problems using IRLS type schemes have been proposed before, see [22, 48].

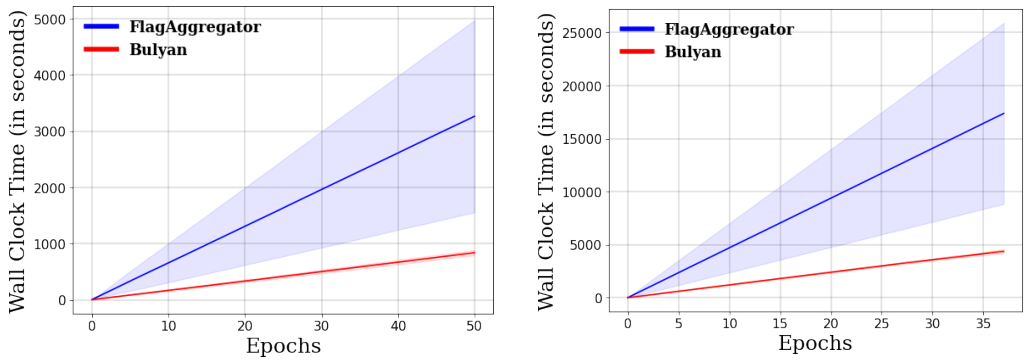
It is well known data augmentation techniques help in improving the generalization capabilities of models by adding more identically distributed samples to the data pool. [49, 50, 51]. The techniques have evolved along with the development of the models, progressing from the basic ones like rotation,

translation, cropping, flipping, injecting Gaussian noise [52] etc., to now the sophisticated ones (random erasing/masking [53], cutout [54] etc.). Multi-modal learning setups [55, 56, 57], use different ways to combine data of different modality (text, images, audio etc.) to train deep learning networks.

5 Discussion and Limitation

Is it possible to fully "offload" FA computation to switches? Recent work propose that aggregation be performed entirely on network infrastructure to alleviate any communication bottleneck that may arise [58, 59]. However, to the best of our knowledge, switches that are in use today only allow limited computation to be performed on gradient g_i as packets whenever they are transmitted [60, 61]. That is, *programmability* is restrictive at the moment— switches used in practice have no floating point, loop support, and are severely memory/state constrained. Fortunately, solutions seem near. For instance, [62] have already introduced support for floating point arithmetic in programmable switches. By adapting some of these ideas, we believe it is possible to implement FA on programmable switches since FA can be computed with Matrix-Vector products. For instance, we can use quantization approaches for SVD calculation with some accuracy loss [63] to approximate floating point arithmetic with fixed point and can use a randomized or streaming implementation of SVD to avoid iterative computations (loops). While we leave the actual implementation and detailed empirical studies of FA on programmable hardware switches for future work, we note that there is no fundamental limitation to the feasibility of FA in real programmable networks. On the positive side, our extensive empirical evidence clearly demonstrate the benefits of using a real implementation of FA on GPUs on large scale settings, and if this computation can be offloaded efficiently, then overall training does indeed get accelerated.

A potential limitation of FA is its computational complexity. Figure 10 show the wall clock time it takes for FA to reach a certain epoch compared to Bulyan, irrespective of the batch size as shown in Figure 10a and Figure 10b. Because in every iteration of FA, we perform SVD, the complexity of the algorithm would be $O(nN_\delta(\sum_{i=1}^{p} k_i)^2)$ with N_δ being the number of iterations for the algorithm.



(a) batch size=200

(b) batch size=25

Figure 10: Wall clock time comparison

6 Conclusion

In this paper we proposed Flag Aggregator (FA) that can be used for robust aggregation of gradients in distributed training. FA is an optimization-based subspace estimator that utilizes Flag Median in modeling pairwise distances as quadratic functions. We perform extensive evaluations of FA and show it can be effectively used in providing byzantine resilience for gradient aggregation. Using techniques from convex optimization, we theoretically analyze FA and with tractable relaxations show its amenability to be solved by off-the-shelf solvers or first-order reweighing methods.

References

[1] Michel Grabisch, Jean-Luc Marichal, Radko Mesiar, and Endre Pap. *Aggregation functions*, volume 127. Cambridge University Press, 2009.

[2] Sivaraman Balakrishnan, Simon S. Du, Jerry Li, and Aarti Singh. Computationally efficient robust sparse estimation in high dimensions. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 2017 Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 169–212. PMLR, 07–10 Jul 2017. URL https://proceedings.mlr.press/v65/balakrishnan17a.html.

[3] Ilias Diakonikolas, Daniel Kane, Sushrut Karmalkar, Eric Price, and Alistair Stewart. Outlier-robust high-dimensional sparse estimation via iterative filtering. *Advances in Neural Information Processing Systems*, 32, 2019.

[4] Yu Cheng, Ilias Diakonikolas, Rong Ge, Shivam Gupta, Daniel M. Kane, and Mahdi Soltanolkotabi. Outlier-robust sparse estimation via non-convex optimization. *Advances in Neural Information Processing Systems*, 2022.

[5] Ilias Diakonikolas, Daniel M. Kane, Sushrut Karmalkar, Ankit Pensia, and Thanasis Pittas. Robust sparse mean estimation via sum of squares. In Po-Ling Loh and Maxim Raginsky, editors, *Proceedings of Thirty Fifth Conference on Learning Theory*, volume 178 of *Proceedings of Machine Learning Research*, pages 4703–4763. PMLR, 02–05 Jul 2022. URL https://proceedings.mlr.press/v178/diakonikolas22e.html.

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[7] Konstantinos I Tsianos and Michael G Rabbat. Distributed strongly convex optimization. In *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 593–600. IEEE, 2012.

[8] Tao Yang, Xinlei Yi, Junfeng Wu, Ye Yuan, Di Wu, Ziyang Meng, Yiguang Hong, Hong Wang, Zongli Lin, and Karl H Johansson. A survey of distributed optimization. *Annual Reviews in Control*, 47:278–305, 2019.

[9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 118–128. Curran Associates Inc., 2017. ISBN 9781510860964.

[10] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/ec1c59141046cd1866bbbcdfb6ae31d4-Paper.pdf.

[11] Leonardo Bautista-Gomez, Ferad Zyulkyarov, Osman Unsal, and Simon McIntosh-Smith. Unprotected computing: A large-scale study of dram raw error rate on a supercomputer. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 645–655, 2016. doi: 10.1109/SC.2016.54.

[12] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, page 1–es, USA, 2007. USENIX Association.

[13] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361, aug 2011. ISSN 0146-4833. doi: 10.1145/2043164.2018477. URL https://doi.org/10.1145/2043164.2018477.

[14] Guosai Wang, Lifei Zhang, and Wei Xu. What can we learn from four years of data center hardware failures? In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 25–36, 2017. doi: 10.1109/DSN.2017.26.

[15] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, Luigi Carro, and Arthur Bland. Understanding gpu errors on large-scale hpc systems and the implications for system design and operation. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–342, 2015. doi: 10.1109/HPCA.2015.7056044.

[16] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H. Rogers. A large-scale study of soft-errors on gpus in the field. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 519–530, 2016. doi: 10.1109/HPCA.2016.7446091.

[17] Sheldon M Ross. *Introduction to probability models*. Academic press, 2014.

[18] Nathan Mankovich, Emily J King, Chris Peterson, and Michael Kirby. The flag median and flagirls. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10339–10347, 2022.

[19] Georgios Damaskinos, El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, and Sébastien Rouault. Aggregathor: Byzantine machine learning via robust gradient aggregation. *Proceedings of Machine Learning and Systems*, 1:81–106, 2019.

[20] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Hervé Lebret. Applications of second-order cone programming. *Linear algebra and its applications*, 284(1-3):193–228, 1998.

[21] Srinadh Bhojanapalli, Nicolas Boumal, Prateek Jain, and Praneeth Netrapalli. Smoothed analysis for low-rank solutions to semidefinite programs in quadratic penalty form. In *Conference On Learning Theory*, pages 3243–3270. PMLR, 2018.

[22] Massimo Fornasier, Holger Rauhut, and Rachel Ward. Low-rank matrix recovery via iteratively reweighted least squares minimization. *SIAM Journal on Optimization*, 21(4):1614–1640, 2011.

[23] Chungen Shen, Yunlong Wang, Wenjuan Xue, and Lei-Hong Zhang. An accelerated active-set algorithm for a quadratic semidefinite program with general constraints. *Computational Optimization and Applications*, 78(1):1–42, 2021.

[24] Ariel Kleiner, Ali Rahimi, and Michael Jordan. Random conic pursuit for semidefinite programming. *Advances in Neural Information Processing Systems*, 23, 2010.

[25] Hans D Mittelmann. An independent benchmarking of sdp and socp solvers. *Mathematical Programming*, 95(2):407–430, 2003.

[26] Robert J Vanderbei and Hande Yurttan. Using loqo to solve second-order cone programming problems. *Constraints*, 1(2), 1998.

[27] Hezhi Luo, Xiaodi Bai, Gino Lim, and Jiming Peng. New global algorithms for quadratic programming with a few negative eigenvalues based on alternative direction method and convex relaxation. *Mathematical Programming Computation*, 11(1):119–171, 2019.

[28] A Shapiro and JD Botha. Dual algorithm for orthogonal procrustes rotations. *SIAM journal on matrix analysis and applications*, 9(3):378–383, 1988.

[29] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[30] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[31] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[33] Rachid Guerraoui, Arsany Guirguis, Jérémy Plassmann, Anton Ragot, and Sébastien Rouault. Garfield: System support for byzantine machine learning (regular paper). In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 39–51, 2021. doi: 10.1109/DSN48987.2021.00021.

[34] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in Byzantium. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3521–3530. PMLR, 10–15 Jul 2018.

[35] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5650–5659. PMLR, 10–15 Jul 2018.

[36] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 571–582, USA, 2014.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[38] David Kelly. Rough path recursions and diffusion approximations. *The Annals of Applied Probability*, 26(1):425–461, 2016.

[39] Jianghong Bao and Qigui Yang. Period of the discrete arnold cat map and general cat map. *Nonlinear Dynamics*, 70(2):1365–1375, 2012.

[40] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=H1oyRlYgg.

[41] Yang You, Jonathan Hseu, Chris Ying, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large-batch training for lstm and beyond. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, 2019.

[42] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching lan speeds. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, page 629–647, 2017.

[43] Jeremy Bernstein, Jiawei Zhao, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd with majority vote is communication efficient and fault tolerant. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[44] Shashank Rajput, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Detox: A redundancy-based framework for faster and more robust gradient aggregation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, 2019.

[45] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. DRACO: Byzantine-resilient distributed training via redundant gradients. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 903–912. PMLR, 10–15 Jul 2018.

[46] Christian Kümmerle, Claudio Mayrink Verdun, and Dominik Stöger. Iteratively reweighted least squares for basis pursuit with global linear convergence rate. *Advances in Neural Information Processing Systems*, 34:2873–2886, 2021.

[47] Deeksha Adil, Richard Peng, and Sushant Sachdeva. Fast, provably convergent irls algorithm for p-norm linear regression. *Advances in Neural Information Processing Systems*, 32, 2019.

[48] Yuxin Chen, Yuejie Chi, Jianqing Fan, Cong Ma, and Yuling Yan. Noisy matrix completion: Understanding statistical guarantees for convex relaxation via nonconvex optimization. *SIAM journal on optimization*, 30(4):3098–3121, 2020.

[49] Fanny Yang, Zuowen Wang, and Christina Heinze-Deml. Invariance-inducing regularization using worst-case transformations suffices to boost accuracy and spatial robustness. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/1d01bd2e16f57892f0954902899f0692-Paper.pdf.

[50] Christina Heinze-Deml and Nicolai Meinshausen. Conditional variance penalties and domain shift robustness, 2017. URL https://arxiv.org/abs/1710.11469.

[51] Saeid Motiian, Marco Piccirilli, Donald A. Adjeroh, and Gianfranco Doretto. Unified deep supervised domain adaptation and generalization. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[52] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995. doi: 10.1162/neco.1995.7.1.108.

[53] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:13001–13008, Apr. 2020. doi: 10.1609/aaai.v34i07.7000. URL https://ojs.aaai.org/index.php/AAAI/article/view/7000.

[54] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[55] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. *International Journal of Computer Vision*, 127:398–414, 2017.

[56] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *ECCV*, 2020.

[57] Yu Huang, Chenzhuang Du, Zihui Xue, Xuanyao Chen, Hang Zhao, and Longbo Huang. What makes multimodal learning better than single (provably). In *NeurIPS*, 2021.

[58] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtárik. Scaling distributed machine learning with {In-Network} aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 785–808, 2021.

[59] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael Swift. {ATP}: In-network aggregation for multi-tenant learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 741–761, 2021.

[60] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 99–110, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320566. doi: 10.1145/2486001.2486011. URL https://doi.org/10.1145/2486001.2486011.

[61] N McKeown. Pisa: Protocol independent switch architecture. In *P4 Workshop*, 2015.

[62] Yifan Yuan, Omar Alama, Jiawei Fei, Jacob Nelson, Dan RK Ports, Amedeo Sapio, Marco Canini, and Nam Sung Kim. Unlocking the power of inline {Floating-Point} operations on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 683–700, 2022.

[63] Zhourui Song, Zhenyu Liu, and Dongsheng Wang. Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

[64] Liqun Qi. Some simple estimates for singular values of a matrix. *Linear algebra and its applications*, 56:105–119, 1984.

A DETAILS OF SECTION 2

We provide the missing details in Section 2 when $m > 1$. To that end, we will assume that each worker i provides the server with a list of k_i gradients, that is, $g_i \in \mathbb{R}^{n\times k}$ – a strict generalization of the case considered in the main paper, useful for our purposes. Note that in [18], these g_i's are assumed to be subspaces whereas we do not make that assumption in our FA algorithm. The advantage of our approach in FA is that: 1. First, in Distributed ML settings, while it is possible to quickly compute stochastic/mini-batch gradients in each worker, for Flag Median, we have to compute the subspace spanned by the mini-batch worker gradient. Unfortunately, this requires at least a QR factorization, which can be expensive to do each training iteration, and moreover 2. Second, in the presence of Byzantine workers, we know that it is crucial to use the norm of the gradients (median) to get provable resilience [34]. So, utilizing the subspace spanned by workers' gradients alone (as done in Flag Median) – ignoring distances – will lead to slow convergence also.

Now, we will show that the RHS in equation (4) and LHS in equation (5) are equivalent. For that, we need to recall an elementary linear algebra fact relating tensor/Kronecker product, and tr operator. Recall the definition of Kronecker product:

Definition A.1. Let $A \in \mathbb{R}^{d_1\times d_2}$, $B \in \mathbb{R}^{e_1\times e_2}$, then $A \otimes B \in \mathbb{R}^{d_1e_1\times d_2e_2}$ is given by,

$$A \otimes B := \begin{bmatrix} a_{1,1}B & \dots & a_{1,d_2}B \\ \vdots & \ddots & \vdots \\ a_{d_1,1}B & \dots & a_{d_1,d_2}B \end{bmatrix}, \tag{6}$$

where $a_{i,j}$ denotes the entry at the $i-$th row, $j-$th column of A.

Lemma A.2 (Equivalence of Objective Functions.)**.** *Let $Y \in \mathbb{R}^{n\times m}$, $g \in \mathbb{R}^{n\times k}$ (so, $M \in \mathbb{R}^{n\times n}$). Then, we have that,*

$$tr\left(Y^T g g^T Y\right) := tr\left(Y^T M Y\right) = \mathbf{vec}(Y)^T \left(I \otimes M\right) \mathbf{vec}(Y), \tag{7}$$

where $I \in \mathbb{R}^{m\times m}$ is the identity matrix.

Proof. Using the definition of tensor product in equation (6), we can simplify the right hand side of equation (7) as,

$$\mathbf{vec}(Y)^T \left(I \otimes M\right) \mathbf{vec}(Y) = [y_{11}, \cdots y_{n1}, \cdots, y_{1m}, \cdots, y_{nm}] \begin{bmatrix} M & 0 & \dots & 0 \\ 0 & M & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & M \end{bmatrix} \begin{bmatrix} y_{11} \\ \vdots \\ y_{n1} \\ \vdots \\ y_{1m} \\ \vdots \\ y_{mn} \end{bmatrix} = \sum_{j=1}^{m} y_j^T M y_j$$

$$= \sum_{j=1}^{m} \text{tr}\left(y_j y_j^T M\right) = \text{tr}\left(\sum_{j=1}^{m} y_j y_j^T M\right) = \text{tr}\left(\left(\sum_{j=1}^{m} y_j y_j^T\right) M\right) \tag{8}$$

$$= \text{tr}\left(Y Y^T M\right) = \text{tr}\left(Y^T M Y\right), \tag{9}$$

where we used the cyclic property of trace operator tr$(\cdot)$ in equations (8), and (9) that is, tr$(ABC) =$ tr$(CAB) =$ tr(BCA) for any dimension compatible matrices A, B, C. □

A.1 Proof of Lemma 2.1

Recall that, given $\tilde{M}_i = I \otimes M_i$, the lifted cone programming relaxation of FA can be written as,

$$\min_{Z} \sum_{i} \sqrt{\text{tr}(Z^T \tilde{M}_i)} \quad \text{s.t.} \quad Z \succeq 0, \;\; \text{tr}(Z) = m, Z = Z^T, \tag{10}$$

where m is the rank of Z or number of columns of Y. We now use the above Lemma A.2 in the supplement to show that the objective function with respect to Z in the lifted formulation is smooth which gives us the desired convergence result in Lemma 2.1.

Lemma A.3 (Restatement of Lemma 2.1)**.** *If L_i are κ_i-smooth, with a η_i-lipschitz Hessian, then projected gradient descent with constant step size converges to a locally optimal solution to* (10) *in $\tilde{O}(\kappa/\epsilon^2)$ iterations where $0 \leq \epsilon \leq \kappa^2/\eta$ is a error tolerance parameter, $\kappa = \max_i \kappa_i$, and $\eta = \max_i \eta_i$. sf*

Proof. To see this, let $\tilde{\kappa}_i > 0$,

$$\frac{\partial \sqrt{\text{tr}(Z^T \tilde{M}_i) + \tilde{\kappa}_i}}{\partial Z} = \frac{1}{2\sqrt{\tilde{\kappa}_i + \text{tr}(Z^T \tilde{M}_i)}} \tilde{M}_i, \tag{11}$$

where $\tilde{M}_i = I \otimes M_i$ as in equation (5). Now, since $\tilde{M}_i$ is constant with respect to Z, the gradient term is affected only through a scalar $\sqrt{\text{tr}(Z^T \tilde{M}_i) + \tilde{\kappa}_i}$. So the largest magnitude or entrywise ℓ_∞-norm of the Hessian is given by,

$$\left| \frac{\partial \frac{1}{\sqrt{\text{tr}(Z^T \tilde{M}_i) + \tilde{\kappa}_i}} \|\tilde{M}_i\|_\infty}{\partial Z} \right| = \frac{\|\tilde{M}_i\|_\infty}{2\sqrt{\left(\text{tr}(Z^T \tilde{M}_i) + \tilde{\kappa}_i\right)^3}}. \tag{12}$$

Now, we will argue that the gradient and hessian are lipschitz continuous in the lifted space. Since any feasible $Z \succeq 0$ is positive semidefinite, if $\tilde{M}_i \succeq 0$, then the scalar $\text{tr}(Z^T \tilde{M}_i)$ is at least $m \cdot \lambda_{mn}^{\tilde{M}_i}$ where $\lambda_{mn}^{\tilde{M}_i}$ is the smallest (or $mn-$th) eigenvalue of $\tilde{M}_i$. So, we can choose $\tilde{\kappa}_i = 0 \forall i$. If not, then there is a negative eigenvalue, possibly repeated. So, the gradient might not exist. In cases where $\tilde{M}_i$ has negative eigenvalues, we can choose $\tilde{\kappa}_i = \tilde{\kappa} = \left|\min_i \min\left(\lambda_{mn}^{\tilde{M}_i}, 0\right)\right|$. With these choices, we have that the gradient of the objective function in (5) is lipschitz continuous. By a similar analysis using the third derivative, we can show that Hessian is also lipschitz continuous with respect to Z. In other words, all the lipschitz constant of both the gradient and hessian of our overall objective function is controlled by $\tilde{\kappa} > 0$. Hence, we have all conditions satisfied required for Lemma 1 in [21], and we have our convergence result for FA in the factored space of $\textbf{vec}(Y)$. □

Algorithm 1 Distributed SGD with proposed Flag Aggregation (FA) at the Parameter Server

Input: Number of workers P, loss functions $f1, f2, ..., f_P$, per-worker minibatch size B, learning rate schedule α_t, initial parameters w_0, number of iterations T

Output: Updated parameters w_T from any worker

for $t = 1$ *to* T **do**
  for $p = 1$ *to* P *in parallel on machine* p **do**
    Select a minibatch: $i_{p,1,t}, i_{p,2,t}, ..., i_{p,B,t}$ $\quad g_{p,t} \leftarrow \frac{1}{B}\sum_{b=1}^{B} \nabla f_{i_{p,b,t}}(w_{t-1})$
  $G_t \leftarrow \{g_{1,t}, \cdots, g_{P,t}\}$ `// Parameter Server receives gradients from all the` P `machines`
  $\hat{Y}_t \leftarrow FA(G_t)$ `// Solve FA in Equation (1) using First Order methods like IRLS at the Parameter Server to obtain` $\hat{Y}$
  Compute update direction as average of columns of $\hat{Y}_t\hat{Y}_t^T G_t$**:** $d_t = \hat{Y}_t\hat{Y}_t^T G_t \mathbf{e}$, where $\mathbf{e} \in \mathbb{R}^P$ is the vector with all entries equal to $1/P$ `// Compute and send` d_t `to all the` P `machines`
  for $p = 1$ *to* P *in parallel on machine* p **do**
    update model: $w_t \leftarrow w_{t-1} - \alpha_t \cdot d_t$
Return w_T

Few remarks are in order with respect to our convergence result. First, **is the choice $\tilde{\kappa}$ important for convergence?** Our convergence result shows that a perturbed objective function $\text{tr}(Z^T \tilde{M}_i) + \tilde{\kappa}$ has the same second order stationary points as that of the objective function in the factored form formulated using Y (or $\textbf{vec}(Y)$). We can avoid this perturbation argument if we explicitly add constraints $\text{tr}(Z^T \tilde{M}_i) \geq 0$, since projections on linear constraints can be performed efficiently

exactly (sometimes) or approximately. Note that these constraints are natural since it is not possible to evaluate the square root of a negative number. Alternatively, we can use a smooth approximate approximation of the absolute values $\sqrt{\left|\text{tr}(Z^T \tilde{M}_i)\right|}$. In this case, it is easy to see from (11), and (12) that the constants governing the lipschitz continuity as dependent on the absolute values of the minimum eigenvalues, as expected. In essence, no, the choice of $\tilde{\kappa}$ does not affect the nature of landscape – approximate locally optimal points remain approximately locally optimal. In practice, we expect the choice of $\tilde{\kappa}$ to affect the performance of first order methods.

Second, **can we assume $\tilde{M}_i \succ 0$ for gradient aggregation purposes?** Yes, this is because, when using first order methods to obtain locally optimal solution, the scale or norm of the gradient becomes a secondary factor in terms of convergence. So, we can safely normalize each M_i by the nuclear norm $\|M_i\|_* := \sum_{j=1}^{k_i} \sigma_j$ where σ_j is the $j-$th singular value of M_i. This ensures that $I - M_i \succeq 0$, assistant convergence. While $\|M_i\|_*$ itself might be computationally expensive to compute, we may be able to use estimates of $\|M_i\|_*$ via simple procedures as in [64]. In most practical implementations including ours, we simply compute the average of the gradients computed by each worker before sending it to the parameter server, that is, $k_i \equiv k = 1$ in which case simply normalizing by the euclidean norm is sufficient for our convergence result to hold. Our FA based distributed training Algorithm 1 solves the factored form for gradient aggregation purposes (in Step 6) at the parameter server.

B ADDITIONAL EXPERIMENTS

B.1 The Effect of Regularization Parameter

Our algorithm depends on the regularization parameter λ. Figure 11 below illustrates the effect of this parameter on similarity of aggregated gradient vectors for FA and MultiKrum. For this experiment, we sample the gradients output by the parameter server across multiple epochs for both FA and MultiKrum and compute the cosine similarity of corresponding vectors. We repeat the experiment with different λ values. As we can see, for smaller iterations there is some similarity between the gradients computed by FA and MultiKrum. This similarity is more visible for smaller λ values.

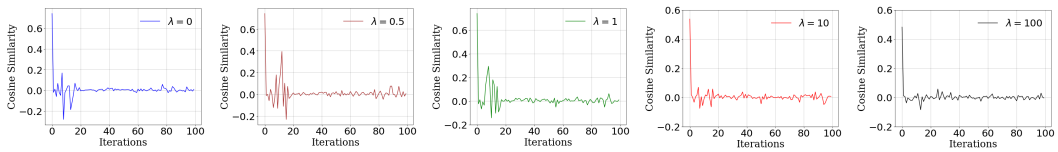


Figure 11: The effect of the regularization parameter λ on similarity of FA performance to MultiKrum

B.2 Experiments with the Tiny ImageNet dataset

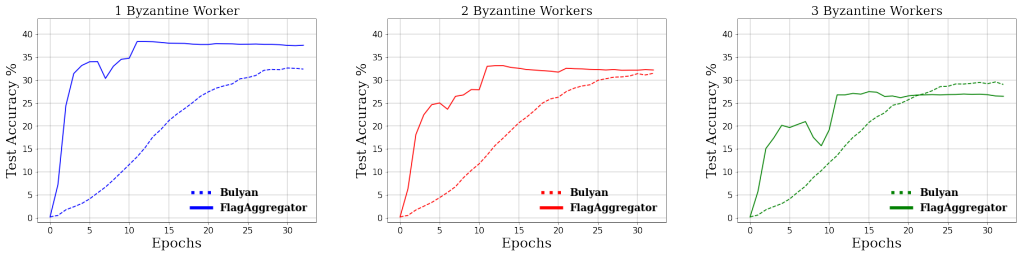
We repeated our experiments with Tiny ImageNet that contains 100000 images of 200 classes (500 for each class) downsized to 64×64 colored images. We fix our batch size to 100 throughout experiments.

Tolerance to the number of byzantine workers: In this experiment we have $n_w = 15$ workers of which $f = 1, .., 3$ are byzantine and send random gradients. The accuracy of test data for FA and Bulyan is shown in Figure 12. As we can see, for $f = 1$ and $f = 2$, FA converges at a higher accuracy. For all cases, FA also converges in 2x less number of iterations compared to Bulyan.

Tolerance to communication loss:

We set a 10% loss rate for the link connecting one of the machines to the parameter server. This machine hosts 5 worker nodes, and subsequently, the loss rate of each one's gradients would be 10%. We compare the results for FA and MultiKrum. Figure 13 shows that our takeaways in the main paper are also confirmed in this setting with the new dataset.

Analyzing the marginal utility of additional workers. In this experiment we fix the number of byzantine workers to $f = 2$ and increase the number of correctly functioning workers with



(a) $f = 1$ (b) $f = 2$ (c) $f = 3$

Figure 12: Tolerance to the number of byzantine workers for robust aggregators for batch size 100.

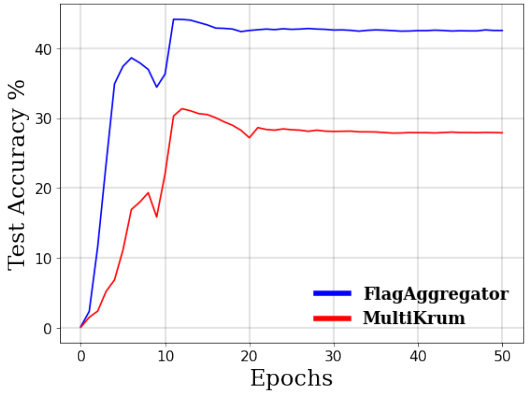
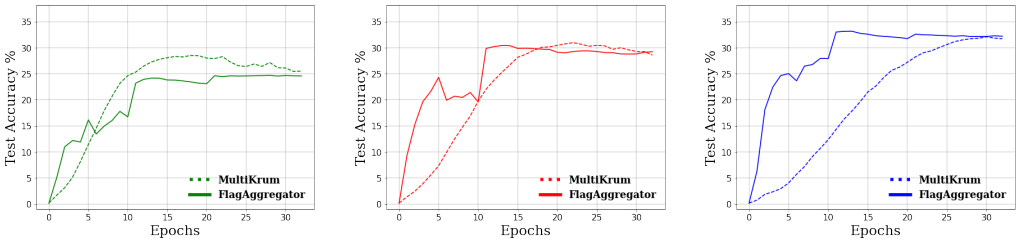


Figure 13: Tolerance to communication loss

TinyImageNet dataset. In Figure 14, we compare the resilience of FA and MultiKrum. As we can see, there are reasonable choices for n_w that FA presents strong resilience similar to MultiKrum.

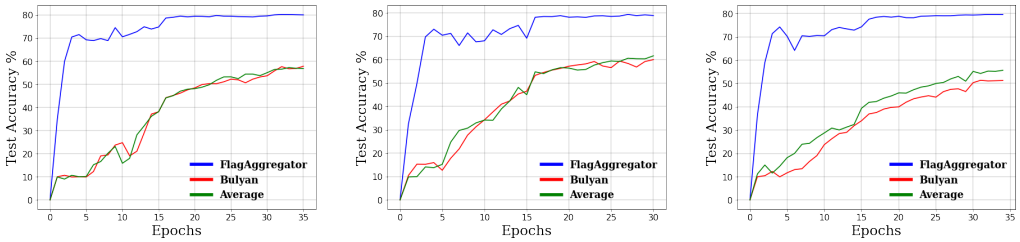


(a) $n_w = 9$ (b) $n_w = 12$ (c) $n_w = 15$

Figure 14: The effect of adding more workers in the presence of fixed number of byzantine workers $f = 2$.

The effect of having augmented data during training in byzantine workers: We choose two non linear augmentation schemes, Lotka Volterra (shown in rows 1 and 3 of Figure 15) and Arnold's Cat Map (shown in rows 2 and 4 of Figure 15). As seen from the figure, Arnold's Cat Map augmentations stretch the images and rearrange them within a unit square, thus resulting in streaky patterns. Whereas the Lotka Volterra augmentations distort the images while keeping the images similar to the original ones. We perform experiments with data augmented with varying shares using the two methods and show the results in Figure 16. The expectation behind these experiments was to see whether the streaky patterns augmented by the Arnold's Cat Map would increase the Byzantine behavior of the workers. Although the results do not show a significant signal, we can see that the performance of all the models was the worst when the proportion of Arnold's Cat Map images was the highest, suggesting that the augmentations did impact the overall gradients. We would like to do further analysis on how different augmentation processes can induce Byzantine behavior, in our future work.

Figure 15: TinyImagenet data with Augmentation: **Row 1**: Lotka Volterra augmentation on Class Horse. **Row 2**: Arnold's Cat Map augmentation on Class Horse. **Row 3**:Lotka Volterra augmentation on Class Ship. **Row 4**: Arnold's Cat Map augmentation on Class Ship.



(a) 100% Lotka-Volterra

(b) 50% Lotka-Volterra, 50% Arnold's Cat Map

(c) 33% Lotka-Volterra, 66% Arnold's Cat Map

Figure 16: Accuracy of using augmented data in $f = 3$ workers